第20章 底层配置

在目标系统上使用μC/GUI 之前，需要为你的应用配置软件。配置指的的是配置（头）文件的修改，这些文件通常位于（子）目录“Config”中。我们尽可能使配置保持简单些，但有一些配置宏（在文件 LCDConf.h 中）是需要的，这是为了使系统能正确工作。包括：

LCD 宏，定义显示屏的尺寸及可选择功能（例如镜像等等）。

LCD 控制宏，定义如何访问你使用的控制器。

20.1 可用的配置宏

下表列出了可供μC/GUI 的底层配置使用的宏：

类型	宏	默认值	说明
通用（必需的）配置			
S	LCD_CONTROLLER	---	选择 LCD 控制器。
N	LCD_BITSPERPIXEL	---	指定每像素的位。
S	LCD_FIXEDPALETTE	---	定义颜色查询表（则必须定义 LCD_PHYSCOLORS）。
N	LCD_XSIZE	---	定义 LCD 的水平分辨率。
N	LCD_YSIZE	---	定义 LCD 的垂直分辨率。
初始化控制器			
F	LCD_INIT_控制器()	---	LCD 控制器初始化顺序。不适用于所有控制器。
显示方向			
B	LCD_MIRROR_X	0	激活 X 轴镜象。
B	LCD_MIRROR_Y	0	激活 Y 轴镜象。
B	LCD_SWAP_XY	0	行转换成列，列转换成行（XY 轴交换）。
N	LCD_VXSIZE	LCD_XSIZE	虚拟显示的水平分辨率。不适用于所有驱动器。
N	LCD_VYSIZE	LCD_YSIZE	虚拟显示的垂直分辨率。不适用于所有驱动器。
N	LCD_XORG<n>	0	LCD controller <n>：最左边（最小的）X 坐标。
N	LCD_YORG<n>	0	LCD controller <n>:最顶端（最小的）Y 坐标。
颜色配置			
N	LCD_MAX_LOG_COLORS	256	在一幅位图中驱动器能支持的逻辑颜色的最大数量。
A	LCD_PHYSCOLORS	---	定义颜色查询表的内容。只有在 LCD_FIXEDPALETTE 被设置为 0 时才需要。
B	LCD_PHYSCOLORS_IN_RAM	0	只有定义了物理颜色的情况下才相应有效。把物理颜色放入 RAM 中，使它们在运行时可以修改。
B	LCD_REVERSE	0	激活在编译时的反转显示。
B	LCD_SWAP_RB	0	激活红、蓝基色的交换。
LCD 的放大			
N	LCD_XMAG<n>	1	LCD 的水平方向放大系数。
N	LCD_YMAG<n>	1	LCD 的垂直方向放大系数。
简单的总线接口配置			
F	LCD_READ_A0(Result)	---	址址线为低电平，从 LCD 控制器读一个字节。
F	LCD_READ_A1(Result)	---	址址线为高电平，从 LCD 控制器读一个字节。
F	LCD_WRITE_A0(Byte)	---	址址线为低电平，向 LCD 控制器写一个字节。
F	LCD_WRITE_A1(Byte)	---	址址线为高电平，向 LCD 控制器写一个字节。
F	LCD_WRITEM_A1	---	址址线为高电平，向 LCD 控制器写入多个字节。
完整总线接口配置			
F	LCD_READ_MEM(Index)	---	读取控制器图像存储器的内容。
F	LCD_READ_REG(Index)	---	读取控制器一个配置寄存器的内容。
F	LCD_WRITE_MEM(Index, Data)	---	向控制器图像存储器（显示数据 RAM）写入内容。

F	LCD_WRITE_REG(Index, Data)	---	向控制器配置寄存器写入内容。
S	LCD_BUSWIDTH	16	选择 LCD 控制器/CPU 的接口的总线宽度(8/16)。
F	LCD_ENABLE_REG_ACCESS	---	切换 M/R 信号进行寄存器访问。并不适用于所有的控制器。
F	LCD_ENABLE_MEM_ACCESS	---	切换 M/R 信号进行存储器访问。并不适用于所有的控制器。
B	LCD_SWAP_BYTE_ORDER	0	在使用一个 16 位总线接口时，激活在 CPU 和 LCD 控制器之间的端模式的反转（高低字节交换）。
LCD 控制器配置：公共极（行）/段（列）连线			
N	LCD_XORG<n>	0	LCD controller <n>：最左边（最小的）X 坐标
N	LCD_YORG<n>	0	LCD controller <n>：最顶端（最小的）Y 坐标
N	LCD_FIRSTSEG<n>	0	LCD controller <n>：使用的第 1 段（列）连线。
N	LCD_LASTSEG<n>	LCD_XSIZE-1	LCD controller <n>：使用的最后 1 段（列）连线。
N	LCD_FIRSTCOM<n>	0	LCD controller <n>：使用的第 1 个公共极（行）连线。
N	LCD_LASTCOM<n>	LCD_YSIZE-1	LCD controller <n>：使用的最后 1 个公共极（行）连线。
COM/SEG（公共极/段） 查询表			
A	LCD_LUT_COM	---	控制器 COM 查询表.
A	LCD_LUT_SEG	---	控制器 SEG 查询表.
杂项			
N	LCD_NUM_CONTROLLERS	1	使用的 LCD 控制器的数量。
B	LCD_CACHE	1	停用以禁止显示数据高速缓存使用，使驱动器的速度慢下来。并不适用于所有的控制器。
B	LCD_USE_BITBLT	1	停用以禁止 BitBLT 引擎，如果设为 1，驱动器将使用所有有效的硬件加速度。
B	LCD_SUPPORT_CACHECONTROL	0	激活以启用驱动器 API 函数 LCD_L0_ControlCache() 的高速缓存控制功能。并不适用于所有的控制器。
N	LCD_TIMERINIT0	---	使用 CPU 作为控制器时，为显示 pane 0 而用于 ISR 的计时值。
N	LCDTIMERINIT1	---	使用 CPU 作为控制器时，为显示 pane 1 而用于 ISR 的计时值。
F	LCD_ON	---	打开 LCD 的函数替换宏。
F	LCD_OFF	---	切断 LCD 的函数替换宏。

如何配置 LCD。

我们推荐使用下面的操作步骤：

1. 对有相似配置的配置文件进行拷贝,我们要对拷贝的文件进行编辑。在目录 Sample\ LCDConf\ xxx 中有几个特定 LCD 控制器的配置范例，其中“xxx”是你的 LCD 驱动

器。

2. 通过定义简单总线或完全总线宏来配置总线接口。
3. 定义你的 LCD 尺寸（LCD_XSIZE, LCD_YSIZE）。
4. 选择你的系统使用的控制器，同时选择合适的 bpp 和调色板模式（LCD_CONTROLLER，LCD_BITSPERPIXEL，LCD_FIXEDPALETTE）。
5. 如果需要，配置所使用的 公共极（行）/段（列）连线。lines。
6. 测试系统。
7. 如果有需要，进行 X/Y 轴反转（LCD_REVERSE）；返回第 6 步。
8. 如果有需要，进行 X/Y 轴镜象（LCD_MIRROR_X, LCD_MIRROR_Y）；返回第 6 步。
9. 检查所有其它配置开关。
10. 删除配置中未使用的部分。

20.2 通用（必需的）配置

LCD_CONTROLLER

描述

定义使用的 LCD 控制器。

类型

选择开关。

附加信息

使用的 LCD 控制器通过适当的数字指定。请参阅第 22 章：“LCD 驱动”以获得更多可用选项的信息。

范例

指定一个 Epson SED1565 控制器：

```
#define LCD_controller 1565      /* 选择 SED1565 LCD 控制器 */
```

LCD_BITSPERPIXEL

描述

指定每像素的位的数量。

类型

数值。

LCD_FIXEDPALETTE

描述

指定固定调色板模式。

类型

选择开关。

附加信息

将数值设置为 0 表示使用一个颜色查询表而不是一个固定设色板模式。这样 LCD_PHYSCOLORS 宏必须定义。

LCD_XSIZE; LCD_YSIZE

描述

（分别）定义所用显示屏水平和垂直的分辨率。

类型

数值。

附加信息

数值是逻辑尺寸;X 轴方向指定了所有 LCD 驱动函数的 X 轴方向。通常 X 轴尺寸等于段(列)的数量。

20.3 初始化控制器

LCD_INIT_CONTROLLER()

描述

初始化 LCD 控制器。

类型

函数替换。

附加信息

该宏必须被用户定义以初始化一些控制器。它在驱动函数 LCD_L0_Init() 和 LCD_L0_Reinit()之间执行。请参考你的控制器的资料手册以获得如何初始化你的硬件设备的更多信息。

范例

下面的范例测试一个 Epson SED1565 控制器，它使用一个内部电源调整器。

```
#define LCD_INIT_CONTROLLER()
    \LCD_WRITE_A0(0xe2);    /* 内部复位 */
    \LCD_WRITE_A0(0xae);    /* 显示开/关：关 */
    \LCD_WRITE_A0(0xac);    /* Power save 开始：static indicator off */
    \LCD_WRITE_A0(0xa2);    /* LCD 偏置选择：1/9 */
    \LCD_WRITE_A0(0xa0);    /* ADC 选择：正常 */
    \LCD_WRITE_A0(0xc0);    /* 公共模式：正常 */
    \LCD_WRITE_A0(0x27);    /* 5V 电压调节器：中 */
    \LCD_WRITE_A0(0x81);    /* 进入电子音量模式 */
    \LCD_WRITE_A0(0x13);    /* 电子音量：中 */
    \LCD_WRITE_A0(0xad);    /* Power save 结束：static indicator on */
    \LCD_WRITE_A0(0x03);    /* static indicator 寄存器设置：开（常开）*/
    \LCD_WRITE_A0(0x2F);    /* 电源控制设置：升压器，调节器及跟随器关闭 */
    \LCD_WRITE_A0(0x40);    /* 显示开始行 */
    \LCD_WRITE_A0(0xB0);    /* 显示地址 0 页 */
    \LCD_WRITE_A0(0x10);    /* 显示列地址 MSB */
```

```
\LCD_WRITE_A0(0x00);     /* 显示列地址 LSB */
\LCD_WRITE_A0(0xaf);     /* 显示 开/关：开 */
\LCD_WRITE_A0(0xe3);     /* 空命令
```

20.4 显示方向

LCD_MIRROR_X

描述

反转显示屏的 X 方向（水平）。

类型

二进制开关

0：禁止，X 轴方向未镜像（默认）；1：激活，X 轴方向镜像。

附加信息

如果激活：X -> LCD_XSIZE-1-X。

该宏与 LCD_MIRROR_Y 和 LCD_SWAP_XY 结合，能用于对显示屏任何方向的支持。在改变这个配置开关之前，确认 LCD_SWAP_XY 设为你的应用所需要的值。

LCD_MIRROR_Y

描述

反转显示屏的 Y 方向（垂直）。

类型

二进制开关

0：禁止，Y 轴方向未镜像（默认）；1：激活，Y 轴方向镜像。

附加信息

如果激活：Y -> LCD_YSIZE-1-Y。

该宏与 LCD_MIRROR_X 和 LCD_SWAP_XY 结合，能用于对显示屏任何方向的支持。在改变这个配置开关之前，确认 LCD_SWAP_XY 设为你的应用所需要的值。

LCD_SWAP_XY

描述

交换显示屏水平和垂直的方向。

类型

二进制开关

0：禁止，X-Y 未交换（默认）；1：激活，X-Y 交换。

附加信息

如果设为 0（没有交换），SEG 连线作为列，COM 连线作为行。如果激活：X -> Y。

修改这个开关时，你也必须交换显示屏分辨率 X-Y 设置(使用 LCD_XSIZE 和 LCD_YSIZE)。

LCD_VXSIZE; LCD_VYSIZE

描述

定义虚拟显示的水平和垂直分辨率。

类型

数值。

附加信息

数值是逻辑尺寸；X 方向指定所有 LCD 驱动函数使用的 X 方向。

如果没有使用虚拟显示器，这些数值应该与 LCD_XSIZE，LCD_YSIZE（这些默认设置）一样。

虚拟显示器特性要求硬件支持，同时并不是对所有驱动器都适用。

LCD_XORG<n>; LCD_YORG<n>

描述

（分别的）定义由配置驱动器控制的显示器的水平和垂直的原点。

类型

数值。

附加信息

在一个单显示屏系统，两个宏通常都设为 0（默认数值）。

20.5 颜色配置

LCD_MAX_LOG_COLORS

描述

定义在一幅位图中驱动器支持的颜色的最大数量。

类型

数值（默认值为 256）。

附加信息

如果你使用 4 级灰度 LCD，通常将该值设为 4 足够了。然而，在这种情况下，记住不要试图使用超过 4 种颜色来显示位图。

LCD_PHYSCOLORS

描述

定义颜色查询表的内容，如果要用到的话。

类型

别名。

附加信息

该宏只要求 LCD_FIXEDPALETTE 设为 0。参考颜色选择以获得更多信息。

LCD_PHYSCOLORS_IN_RAM

描述

如果启用的话，把物理颜色的内容放入 RAM 中。

类型

二进制开关

0：停用（默认值）；1：激活。

LCD_REVERSE

描述

在编译时翻转显示屏。

类型

二进制开关

0：停用，不翻转（默认值）；1：激活，翻转。

LCD_SWAP_RB

描述

交换红蓝两种藕色。

类型

二进制开关

0：停用，不交换（默认值）；1：激活，交换。

20.6 LCD 的放大

为了能正确地显示图片，一些硬件要求 LCD 能放大。这样，必须通过软件对这些有放大需要的硬件进行补偿。这可以通过激活一个层（在驱动层之上），自动处理显示屏的放大工作而做到。

LCD_XMAG

描述

指定 LCD 水平放大系数。

类型

数值（默认为 1）。

附加信息

因数为 1 表示没有放大。

LCD_YMAG

描述

指定 LCD 垂直放大系数。

类型

数值（默认是 1）

附加信息

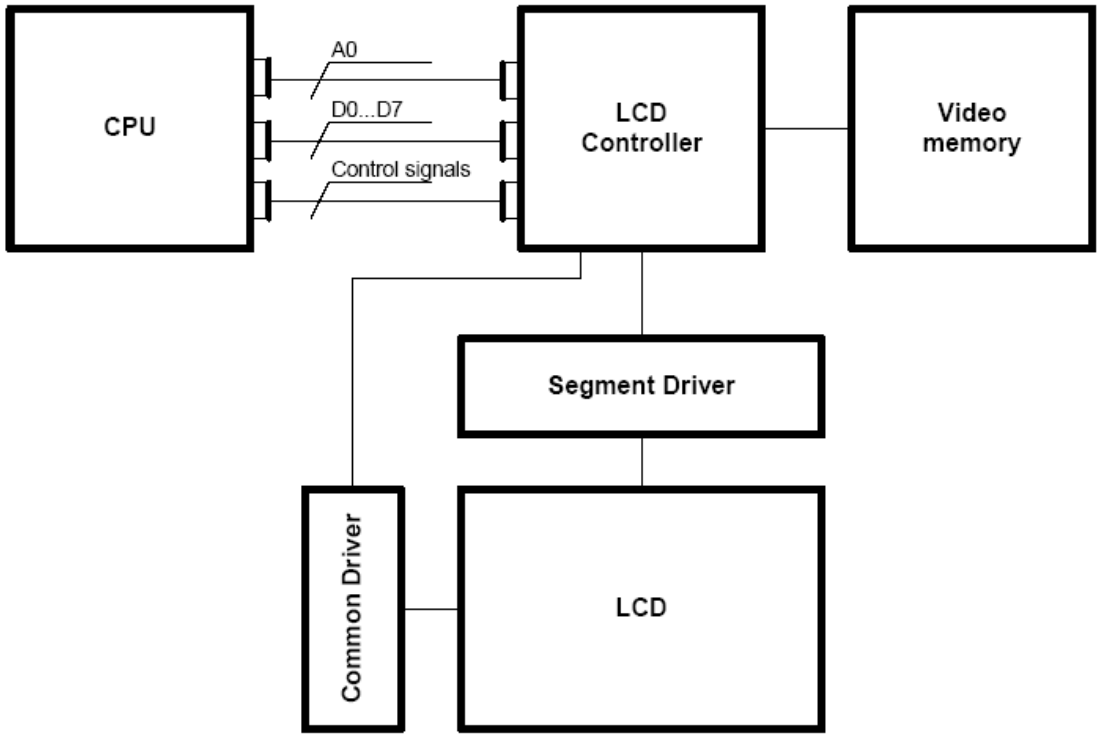
因数为 1 表示没有放大。

20.7 简单总线接口配置

LCD 控制器有两种基本类型的总线接口：完全和简单的总线接口。

大多数用于小一些的显示屏（通常最大为 240×128 或 320×240）的 LCD 控制器使用一个简单总线接口与 CPU 连接。对于一个简单总线，只有一个地址位（通常是 A0）连接到 LCD 控制器。一些这样的控制器非常慢，所以硬件设计者可以考虑将其连接到 I/O 脚而不是地址总线。

带有简单总线的 LCD 控制器的方框图



8 个数据位，一个地址位和 2 或 3 根控制线连接 CPU 和 LCD 控制器。4 个宏告诉 LCD 驱动器如何访问每个使用的控制器。如果 LCD 控制器直接连接到 CPU 的地址总线，配置就简单了，通常每个宏只占用一根连线。如果 LCD 控制器连到 I/O 脚，必须要模拟总线接口，每个宏要占用大约 5～10 根连线（或者调用一个模拟总线接口的函数）。

下面的宏仅用于简单总线接口的 LCD 控制器。

LCD_READ_A0

描述

在 A0（C/D）引脚为低电平时，从 LCD 控制器读取一个字节。

类型

函数替换。

原型

```
#define LCD_READ_A0(Result)
```

参数	含义
Result	读取的结果。这不是一个指针，而是一个变量（读取的数值保存在里面）的占位符。

LCD_READ_A1

描述

在 A0（C/D）引脚为低电平时，从 LCD 控制器读取一个字节。

类型

函数替换。

原型

```
#define LCD_READ_A1(Result)
```

参数	含义
Result	读取的结果。这不是一个指针，而是一个变量（读取的数值保存在里面）的占位符。

LCD_WRITE_A0

描述

在 A0（C/D）引脚为低电平时，向 LCD 控制器写一个字节。

类型

函数替换。

原型

```
#define LCD_WRITE_A0(Byte)
```

参数	含义
Byte	写入的字节。

LCD_WRITE_A1

描述

在 A0（C/D）引脚为高电平时，向 LCD 控制器写一个字节。

类型

函数替换。

原型

```
#define LCD_WRITE_A1(Byte)
```

参　数	含　意
Byte	写入的字节。

LCD_WRITEM_A1

描述

在 A0（C/D）引脚为高电平时，向 LCD 控制器写入几个字节。

类型

函数替换。

原型

```
#define LCD_WRITEM_A1(paBytes, NumberOfBytes)
```

参　数	含　意
paBytes	第一个字节数据指针的占位符。
NumberOfBytes	要写入的数据的字节数。

实际总线接口范例

下面的范例展示如何通过一个实际的总线接口访问：

```
void WriteM_A1(char *paBytes, int NummerOfBytes)
{
    int i;
    for(i = 0; i < NummerOfBytes; i++)
    {
        (*(volatile char * ) 0xc0001) = *(paBytes + i) ;
  }
}
#define  LCD_READ_A1(Result)        Result =(*(volatile char *)0xc0000)
#define  LCD_READ_A0(Result)        Result =(*(volatile char *)0xc0001)
#define  LCD_WRITE_A1(Byte)     (*(volatile char *)0xc0000) = Byte
#define  LCD_WRITE_A0(Byte)         (*(volatile char *)0xc0001) = Byte
#define  LCD_WRITEM_A1(paBytes,NummerOfBytes)
                                    WriteM_A1(  paBytes, NummerOfBytes)
```

连接到I/O 脚的简单函数

在目录 Sample\LCD_X 下可以找到的几个范例：

- 6800 接口的端口函数。
- 8080 接口的端口函数。
- 一个串行接口的简单端口函数。
- 一个简单I^2C总线接口的端口函数。

这些范例可以直接使用。你需要做的是定义在每个范例顶部的端口访问宏，并将它们映射到你的 LCDConf.h 中，与下面展示的样式类似：

```
void LCD_X_Write00(char c);
void LCD_X_Write01(char c);
char LCD_X_Read00(void);
char LCD_X_Read01(void);
#define LCD_WRITE_A1(Byte)      LCD_X_Write01(Byte)
#define LCD_WRITE_A0(Byte)      LCD_X_Write00(Byte)
#define LCD_READ_A1(Result)     Result = LCD_X_Read01()
```

```
#define LCD_READ_A0(Result)      Result = LCD_X_Read00()
```

注意不是所有的 LCD 控制器用同样的方法处理 A0 或 C/D 位。例如，一个 Toshiba 控制器要求在存取数据时，该位为低电平，而 Epson SED1565 要求它为高电平。

多LCD 控制器的硬件访问

如果 LCD 使用的 LCD 控制器数量超过一个，你必须根据你的硬件要求为它们分别单独定义存取宏。每个 LCD 控制器需要 4 个宏。附加控制器的宏常常与第一个控制器的宏非常相似。对于总线的直接连接的情况，通常只是地址不同的。而使用 I/O 引脚连接时，除了片选信号以外，其它存取时序是一样的。

当使用的 LCD 控制器数量超过一个时，控制器的宏要增加一条下划线和控制器的索引作为后缀。例如：

控制器 #0: LCD_READ_A0_0 ，控制器 #1: LCD_READ_A0_1 等等。

注意第 1 个控制器被当作是控制器 #0，这样第 2 个控制器定义为#1，等等。附加 LCD 控制器的宏在下表列出。

第 2 个 LCD 控制器

类型	宏	说明
F	LCD_READ_A0_1(Resu lt)	LCD 控制器 1: A0 = 0 时读一个字节。
F	LCD_READ_A1_1(Result)	LCD 控制器 1: A0 = 1 时读一个字节。
F	LCD_WRITE_A0_1(Byte)	LCD 控制器 1: A0 = 0 时写一个字节。
F	LCD_WRITE_A1_1(Byte)	LCD 控制器 1: A0 = 1 时写读一个字节。

第 3 个 LCD 控制器

类型	宏	说明
F	LCD_READ_A0_2(Resu lt)	LCD 控制器 2: A0 = 0 时读一个字节。
F	LCD_READ_A1_2(Result)	LCD 控制器 2: A0 = 1 时读一个字节。
F	LCD_WRITE_A0_2(Byte)	LCD 控制器 2: A0 = 0 时写一个字节。
F	LCD_WRITE_A1_2(Byte)	LCD 控制器 2: A0 = 1 时写读一个字节。

第 4 个 LCD 控制器

类型	宏	说明
F	LCD_READ_A0_3(Resu lt)	LCD 控制器 3: A0 = 0 时读一个字节。

F	LCD_READ_A1_3(Result)	LCD 控制器 3: A0 = 1 时读一个字节。
F	LCD_WRITE_A0_3(Byte)	LCD 控制器 3: A0 = 0 时写一个字节。
F	LCD_WRITE_A1_3(Byte)	LCD 控制器 3: A0 = 1 时写读一个字节。

20.8 完全总线接口配置

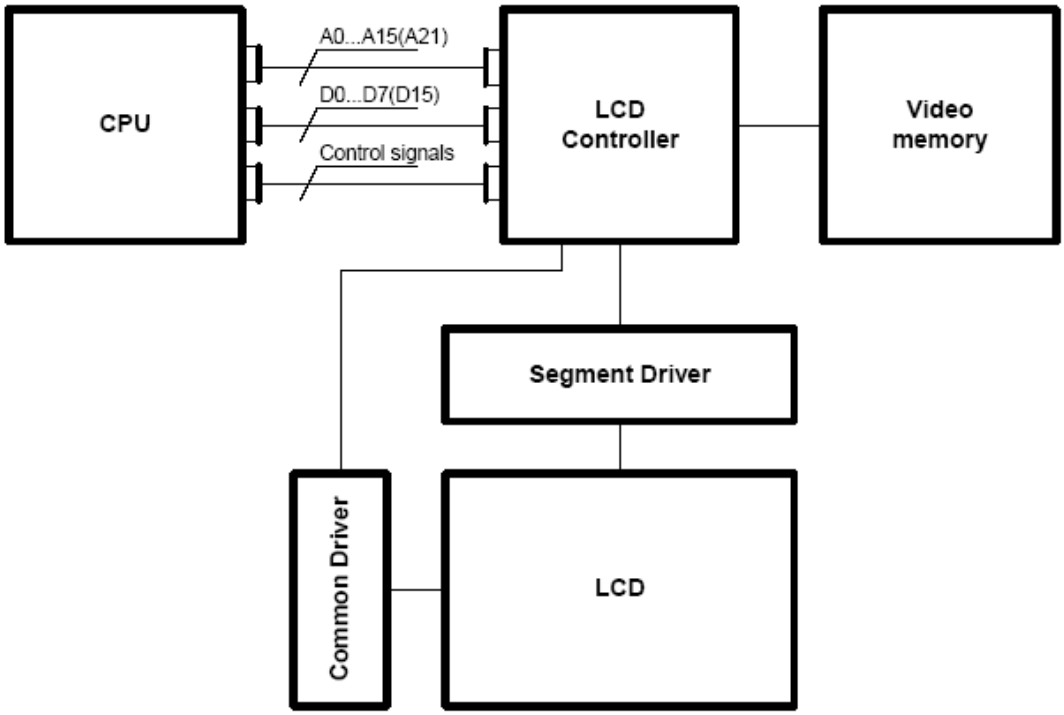
一些 LCD 控制器（特别是那些用于更高分辨率显示的显示屏的控制器）要求一个完全地址总线，意思是说它们至少需要要连接 14 个地址位。在一个完全地址总线配置中，CPU 直接访问图像存储器；完全地址总线连接到 LCD 控制器。

配置一个完全地址总线接口时唯一要知道的是地址范围（产生 LCD 控制器的片选信号）以及是使用 8 位还是 16 位存取（LCD 控制的地址总线宽度）。换句话说，你需要知道下面这些内容：

- 图像存储器存取的基地址；
- 寄存器访问的基地址；
- 相邻的图像存储器位置之间的距离（通常是 1/2/4 字节）；
- 相邻的寄存器位置之间的距离（通常是 1/2/4 字节）；
- 图像存储器的存取类型（8/16/32 位）。

·寄存器的存取类型（8/16/32 位）

完全总线接口的LCD 控制器的典型框图



配置范例

该范例假设以下条件成立：

图像存储器基地址	0x80000
寄存器基地址	0xc0000
视频 RAM 存取	16 位
寄存器存取	16 位
相邻的图像存储器位置之间的距离	2 字节
相邻的寄存器位置之间的距离	2 字节

```
#define LCD_READ_REG(Index)              *((U16* )(0xc0000+(Off<<1)))
#define LCD_WRITE_REG(Index,data)        *((U16*)(0xc0000+(Off<<1)))=data
#define LCD_READ_MEM(Index)              *((U16* )(0x80000+(Off<<1)))
#define LCD_WRITE_MEM(Index,data)        *((U16*)(0x80000+(Off<<1)))=data
```

下面的宏仅用于带有完全总线接口的 LCD 控制器。

LCD_READ_MEM

描述

从 LCD 控制器的图象存储器读取数据。

类型

函数替换。

原型

```
#define LCD_READ_MEM(Index)
```

参　数	含　意
Index	控制器图像存储器的索引。

附加信息

该宏定义如何读取 LCD 控制器的图像存储器。

为了能正确配置这个开关，你需要知道图像存储器的基地址，间隔及是否允许 8/16/或 32 位存取。你也应知道适合你的编译器的正确的语法，因为这种硬件存取不是基于 ANSI C 定义的，因此会因为不同的编译器而不一样。

LCD_READ_REG

描述

从 LCD 控制器的寄存器读取数据。

类型

函数替换。

原型

```
#define LCD_READ_REG(Index)
```

参　数	含　意
Index	读取的寄存器的索引。

附加信息

该宏定义如何读取 LCD 控制器的寄存器。通常寄存器是内存映射的。在这种情况下，该宏一般可以单独写为一行。

为了能正确配置这个开关，你需要知道寄存器映射到的的地址，间隔及是否允许 8/16/或 32 位存取。你也应知道适合你的编译器的正确的语法，因为这种硬件存取不是基于 ANSI C 定义的，因此会因为不同的编译器而不一样。不过，下面的语法可以在它们当中大部分正常工作。

范例

如果寄存器映射至起始地址为 0xc0000 的内存区，间隔为 2 以及使用 16 位存取方式；对于大部分编译器，定义看起来应该像下面一样：

```
#define LCD_READ_REG(Index)      *((U16*)(0xc0000+(Off<<1)))
```

LCD_WRITE_MEM

描述

向 LCD 控制器的图像存储器写数据。

类型

函数替换。

原型

```
LCD_WRITE_MEM(Index, Data)
```

参　数	含　意
Index	控制器的图像存储器的索引。

附加信息

该宏定义如何向 LCD 控制器的图像存储器写数据。

为了能正确配置这个开关，你需要知道图像存储器的基地址，间隔和是否允许 8/16 或者

32 位存取，还有适合你的编译器的正确语法。

对于 8 位存取，数值 1 表示 1 个字节。对于 16 位存取，数值 1 表示 1 个字。

LCD_WRITE_REG

描述

向 LCD 控制器指定的寄存器写入数据。

类型

函数替换。

原型

```
LCD_WRITE_REG(Index, Data)
```

参　数	含　意
Index	写入的寄存器的索引。

附加信息

该宏定义如何写 LCD 控制器的寄存器。如果寄存器是内存映射的，该宏一般可以单独写作一行。为了能正确配置这个开关，你需要知道寄存器映射到的地址，间隔和是否允许 8/16 或者 32 位存取，还有适合你的编译器的正确语法。

对于 8 位存取，数值 1 表示 1 个字节。对于 16 位存取，数值 1 表示 1 个字。

范例

如果寄存器映射至起始地址为 0xc0000 的内存区，间隔为 4 以及使用 8 位存取方式； 对于大部分编译器来说，定义看起来应该像下面一样：

```
#define     LCD_WRITE_REG(Index,Data)
                              *((U8volatile *)(0xc0000+(0ff<<2)))=data
```

LCD_BUSWIDTH

描述

定义 LCD 控制器/CPU 接口（外部显示屏访问）的总线宽度。

类型

选择开关。

8：8 位宽度 VRAM；16：16 位宽度 VRAM （默认）

附加信息

因为这些完全依赖于你的硬件，你将不得不替换这些宏。Epson SED1352 在存储器和寄存器之间是有区别的；存储器是 LCD 控制器的图像存储器，寄存器是 15 个配置寄器。该宏定义如何存取（读/写）VRAM 和寄存器。

LCD_ENABLE_REG_ACCESS

描述

启用寄存器访问并将 M/R 信号设置为高。

类型

函数替换。

原型

```
#define     LCD_ENABLE_REG_ACCESS() MR = 1
```

附加信息

只用于 Epson SED1356 和 SED1386 控制器。

使用该宏之后，LCD_ENABLE_MEM_ACCESS 也必须被定义，为了在寄存器访问后切换回存储器访问。

LCD_ENABLE_MEM_ACCESS

描述

切换 M/R 信号到存储器访问。它在寄存器访问函数后执行，并将 M/R 信号设置为低。

类型

函数替换。

原型

```
#define     LCD_ENABLE_MEM_ACCESS() MR = 0
```

附加信息

只用于 Epson SED1356 及 SED1386 控制器。

LCD_SWAP_BYTE_ORDER

描述

使用 16 位地址总线时，在 CPU 和 LCD 控制器中反转端模式（交换高低字节）。

类型

二进制开关

0：停用，端模式没有交换（默认）；1：激活，端模式交换。

20.9 LCD 控制器配置：公共极（行）/段（列）连线

对于大部分 LCD，公共极（COM）和段（SEG）连线的设置是易于理解的，既不需要特定的设置，也不需要配置宏。

这一部分说明 LCD 控制器如何与你的显示屏进行物理的连接。方向无关紧要，它只是假设 COM 和 SEG 连线是连续的。如果 SEG 或 COM 的方向反了，使用 LCD_MIRROR_X/LCD_MIRROR_Y 设置它们为你的应用所需要的方向。如果使用非连续的 COM/SEG 连线地，你必须修改驱动器（插入一个翻译表）或者最好找到硬件（LCD 模块）设计者，告诉他/她要重新设计。

LCD_XORG<n>；LCD_YORG<n>

描述

通过配置驱动器，（分别）定义显示屏水平和垂直方向的原点。

类型

数值。

附加信息

在一个单显示屏系统，两个宏通常都设为 0（默认值）。

LCD_FIRSTSEG<n>

描述

Controller <n>：使用的第 1 段（列）连线。

类型

数值。

LCD_LASTSEG<n>

描述

Controller <n>：使用的最后 1 段（列）连线。

类型

数值。

LCD_FIRSTCOM<n>

描述

Controller <n>：使用的第1个公共极（行）连线。

类型

数值。

LCD_LASTCOM<n>

描述

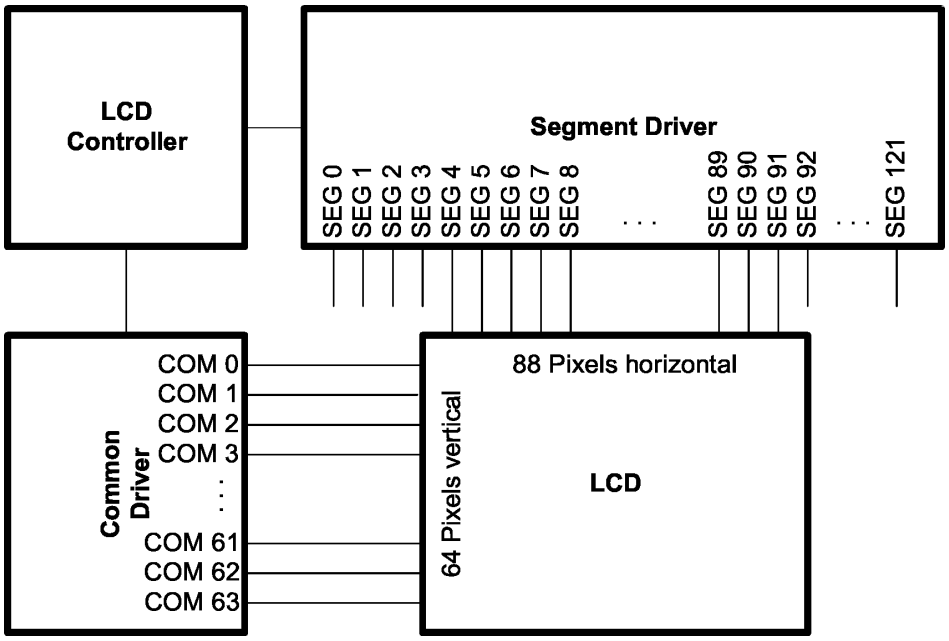
Controller <n>：使用的最后 1 个公共极（行）连线。

类型

数值。

单LCD 控制器配置

下面的框图展示由单个 LCD 控制器控制的单个 LCD。使用另外的 COM 和 SEG 驱动器。所有公共极驱动器的输出口（COM0～COM63）都用上， 但是只有部分段驱动器的输出口（SEG4～SEG91）被用到。注意：为了简明起见，视频 RAM 没有在框图内给出。



上面范例的配置

```
#define     LCD_FIRSTSEG0    (4)     /* Contr.0: 使用的第 1 条 seg 连线 */
#define     LCD_LASTSEG0     (91)    /* Contr.0: 使用的最后 1 条 seg 连线 */
#define     LCD_FIRSTCOM0    (0)     /* Contr.0: 使用的第 1 条 com 连线 */
#define     LCD_LASTCOM0     (63)    /* Contr.0: 使用的最后 1 条 com 连线 */
```

同时也请注意，如果 COM 或 SEG 连线被镜象，甚至 LCD 侧转（90° 旋转，X、Y 轴互换），上面的配置也是同样的。如果 COM/SEG 驱动器集成到 LCD 控制器当中，如一些为小 LCD 所设

计的控制器方案一样，也同样适用。这种控制器的一个典型的例子是 Epson SED15XX 系列。

配置附加 LCD 控制器

µC/GUI 提供了一个 LCD 可以使用多个 LCD 控制器（最多为 4 个）的可能。配置开关与第一个控制器(控制器 0)是一样的，除了索引是 1，2 或 3，而不是 0。

第 2 个 LCD 控制器

类型	宏	说明
N	LCD_FIRSTSEG1	LCD 控制器 1: 使用的第 1 段连线。
N	LCD_LASTSEG1	LCD 控制器 1: 使用的最后 1 段连线。
N	LCD_FIRSTCOM1	LCD 控制器 1: 使用的第 1 个公共极连线。
N	LCD_LASTCOM1	LCD 控制器 1: 使用的最后 1 个公共极连线。

第 3 个 LCD 控制器

类型	宏	说明
N	LCD_FIRSTSEG2	LCD 控制器 2: 使用的第 1 段连线。
N	LCD_LASTSEG2	LCD 控制器 2: 使用的最后 1 段连线。
N	LCD_FIRSTCOM2	LCD 控制器 2: 使用的第 1 个公共极连线。
N	LCD_LASTCOM2	LCD 控制器 2: 使用的最后 1 个公共极连线。

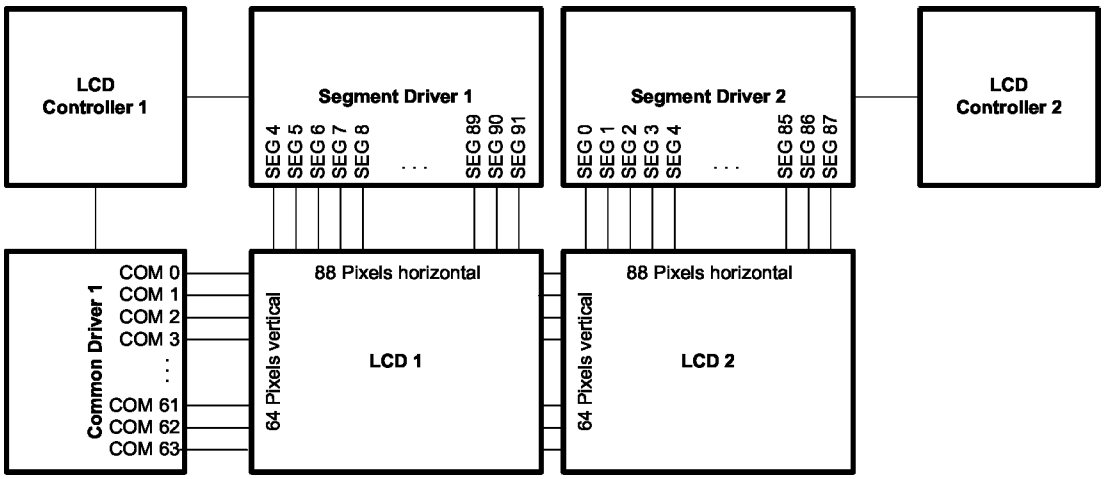
第 4 个 LCD 控制器

类型	宏	说明
N	LCD_FIRSTSEG3	LCD 控制器 3: 使用的第 1 段连线。
N	LCD_LASTSEG3	LCD 控制器 3: 使用的最后 1 段连线。
N	LCD_FIRSTCOM3	LCD 控制器 3: 使用的第 1 个公共极连线。
N	LCD_LASTCOM3	LCD 控制器 3: 使用的最后 1 个公共极连线。

当使用 LCD 控制器数量超过一个时，记住要确定定义了所使用的控制器的数量（参考下一节的宏 LCD_NUM_CONTROLLERS ）。

范例

下图展示了一个使用两个 LCD 控制器的硬件配置。COM 连线由公共极驱动器驱动连接到控制器 1，并且直接连接到第 2 个 LCD。LCD 1 连接到段驱动器 1，使用 SEG 连线 4 到 91。LCD 2 通过段驱动器 2 的 SEG 0 到 SEG 87 驱动。



上面范例的配置

```
#define     LCD_FIRSTSEG0   (4)     /* Contr.0：使用的第 1 条 seg 连线。 */
#define     LCD_LASTSEG0    (91)    /* Contr.0：使用的最后 1 条 seg 连线。 */
#define     LCD_FIRSTCOM0   (0)     /* Contr.0：使用的第 1 条 com 连线。*/
#define     LCD_LASTCOM0    (63)    /* Contr.0： 使用的最后 1 条 com 连线。 */
#define     LCD_XORG0       (0)     /* Contr.0：最左边（最小）X 坐标 */
#define     LCD_YORG0       (0)     /* Contr.0: 最顶端（最小）Y 坐标 */
#define     LCD_FIRSTSEG1   (0)     /* Contr.1：使用的第 1 条 seg 连线 */
#define     LCD_LASTSEG1    (87)    /* Contr.1：使用的最后 1 条 seg 连线。 */
#define     LCD_FIRSTCOM1   (0)     /* Contr.1：使用的第 1 条 com 连线。 */
#define     LCD_LASTCOM1    (63)    /* Contr.1： 使用的最后 1 条 com 连线。 */
#define     LCD_XORG1       (88)    /* Contr.1: 最左边（最小）X 坐标 */
#define     LCD_YORG1       (0)     /* Contr.1: 最顶端（最小）Y 坐标 */
```

20.10 COM/SEG 查询表

当使用“chip on glass”技术时，有时非常难保证控制器的 COM 和 SEG 输出是以线性方式连接到情况显示屏。在这种情况下，要求有一个 COM/SEG 查询表，告诉驱动器关于 COM/SAEG 连线是如何连接。

LCD_LUT_COM

描述

为控制器定义一个 COM 查询表。

类型

别名。

范例

我们假设你的显示屏只包含 10 条 COM 连线，它们的连接序列是 0，1，2，6，5，4，3， 7，8,9。为了配置 LCD 驱动器，使 COM 连线能以正确的序列访问，下面的宏应该加入你的 LCDConf.h 文件当中：

```
#define      LCD_LUT_COM      0, 1, 2, 6, 5, 4, 3, 7, 8, 9
```

如果你需要修改段序列，你应该以同样的形式使用 LCD_LUT_SEG 宏。

LCD_LUT_SEG

描述

为控制器定义一个 SEG 查询表。

类型

别名。

20.11 杂项

LCD_NUM_CONTROLLERS

描述

定义使用的 LCD 控制器数量。

类型

数值（默认为 1）。

LCD_CACHE

描述

控制 CPU 内存中视频内存的高速缓存。

类型

二进制开关

0：禁止，没有使用显示数据高速缓存；1：启用，使用显示数据高速缓存（默认）。

附加信息

该开关并不是所有 LCD 驱动器都支持。

如果访问图像存储器的速度太慢的话，推荐使用一个显示屏高速缓存（可以对存取速度进行加速），这是大一些的显示屏和使用简单总线接口（特别是使用端口访问或串行接口）时的通常做法。禁止高速缓存会使用驱动器的速度慢下来。

LCD_USE_BITBLT

描述

控制硬件加速度的使用。

类型

二进制开关。

0：禁止，未使用 BitBLT 引擎；1：启用，使用 BitBLT 引擎（默认）。

附加信息

禁止 BitBLT 引擎将通知驱动器不使用有效的硬件加速。

LCD_SUPPORT_CACHECONTROL

描述

开关支持驱动器函数 LCD_L0_ControlCache()。

类型

二进制开关

0：禁止，LCD_L0_ControlCache() 不能使用（默认）；1：启用，LCD_L0_ControlCache() 可以使用。

附加信息

API 函数 LCD_L0_ControlCache() 允许高速缓存的锁定，解锁或清除。要了解更多的信息，参阅第 23 章“LCD 驱动 API 函数”。请注意，该特性只有一些使用简单总线接口的 LCD 控制器才有。用尽可能短的时间访问控制器这很更要，这样可以获得最大的速度。对于其它控制器，该开关无效。

LCD_TIMERINIT0

描述

用于一个显示一个像素的 pane 0 的中断服务函数的定时值。

类型

数值。

附加信息

该宏只有在没有使用 LCD 控制器时才有作用，因为它是一项 CPU 通过一个中断服务程序来更新显示的工作。

LCD_TIMERINIT1

描述

用于一个显示一个像素的 pane 1 的中断服务函数的定时值。

类型

数值。

附加信息

该宏只有在没有使用 LCD 控制器时才有作用，因为它是一项 CPU 通过一个中断服务程序

来更新显示的工作。

LCD_ON

描述

切换 LCD 到开状态。

类型

函数替换。

LCD_OFF

描述

切换 LCD 到关状态。

类型

函数替换。